Procedural Generation using Grammar based Modelling and Genetic Algorithms

Karl Haubenwallner*
Supervised by: Markus Steinberger†

Institute of Computer Graphics
Graz University of Technology / Austria

Procedural modeling with shape grammars is a powerful tool to create complex 3D models, but the results are often difficult to control. In this paper we investigate the use of Genetic Algorithms as an optimization algorithm to find a suitable solution for a given target shape. We present a genome representation, a crossover-operator and mutation operators for shape grammars. Furthermore, we demonstrate the feasibility of this approach, using a grammar for spaceships and a volumetric evaluation method, and evaluate the parameters for the genetic algorithm.

Keywords: Procedural Generation, Genetic Algorithms, Shape Grammar, Computer Graphics

1 Introduction

The last years have seen an increasing use of vast open worlds in games such as *The Elder Scrolls* series, the *Fallout* series and *Grand Theft Auto*, among others. Such worlds can increase the immersion during gameplay immensely, but creating such large worlds is a time-consuming and mostly tedious task for artists.

Procedural generation and shape grammars offer the possibility to create arbitrarily large and complex worlds and models algorithmically from a small set of rules, thus allowing artists and designers to focus on the narrative and compelling aspect of those worlds. Unfortunately shape grammars are notoriously difficult to control, and small changes in the rules can produce huge differences in the outcome, which only changes the task of creating the world to the equally time-consuming task of finding the right rules and parameters.

Recently there has been some progress in using various methods from machine learning to control the result of procedural generation, where an algorithm takes a shape grammar and a high-level specification, e.g. a sketch or volumetric shapes, as input and generates a derivation tree for the grammar to produce a model that best matches the specification.

While existing work mostly uses Markov Chain Monte Carlo (MCMC) methods and variations thereof, we present an alternative method using Genetic Algorithms (GA), which have the advantage of creating many equally viable solutions, and therefore providing access to a creative solution process.

1.1 Shape Grammars

One possible approach when creating 3D models is to start with a basic shape (e.g. a cube) and deform and modify it until it resembles the desired model. This is done by applying various operations to the basic shape like translation and rotation or more involved ones like extrusion or splitting, which results in a complex model that bears only a slight resemblance to the initial shape.

Shape grammars try to codify this approach by defining an initial state (called *axiom*), assigning *symbols* to the shapes and defining *production rules*, which declare how to generate symbols and how apply the different operations. This allows them to define a complex model with an initial axiom and a sequence of production rules only. By passing the axiom and the rules to a production system, which applies the operations defined in the rules, an actual model is generated. Since the operations can create new shapes, and the resulting model depends on the sequence of operations applied to certain shapes, the sequence is usually stored as a tree structure, called a *derivation tree*.

1.2 Genetic Algorithms

Genetic Algorithms, a subset of Evolutionary Algorithms (EA), are iterative optimization algorithms with the ability to generate many different, equally viable solutions for any given problem, and therefore provide access to a creative solution process.

The method is inspired by evolution and natural selection, where traits and characteristics of individuals of a species are encoded as genes and chromosomes, and individuals with successful traits get more chances to pass on their genes to future generations, while less successful traits tend to disappear, thus leading to a better adapted population.

GAs follow this process by introducing a *genome representation*, which is a indirect encoding of the problem space. These genes, traditionally symbols of a bit-string,

*karl.haubenwallner@student.tugraz.at
†markus.steinberger@mpi-inf.mpg.de

are then assembled into a *chromosome*, which form an *individual* that represents one possible solution for a given problem. They then go on to create a random set of individuals, which form the initial population, and explore the problem space by evaluating the individuals and assigning each a fitness value, and by selecting the fittest individuals and combining them to form a new generation of possible solutions, which in turn serve as the parents for the next generation.

2 Related work

Shape Grammars Shape grammars were first introduced by Stiny [20], and Lindenmayer [7] used them to create plant models using algorithms. They were expanded with various operators by Stiny [21] and Wonka et. al. [23] among others. Muller et al. [9] introduced the shape grammar *CGA Shape* to generate architectural models on a large scale by iterative refining shapes from a basic vocabulary, and Schwarz et al [15] expanded *CGA Shape* with *CGA++* by introducing boolean operators, and simultaneous operations on groups of shapes.

Procedural modeling There have been several works with procedural modeling using shape grammars, such as generating road networks [12], or generating and rendering architecture and cities on the GPU [6, 19], and several works using inverse procedural modeling, such as recreating trees with biological models [18], or creating derivation trees for shape grammars using MCMC methods [14, 22], or using constraint systems [8].

Genetic Algorithms and evolutionary computing GAs were introduced by Holland in [5], and in turn have been adapted to a wide range of problems, several of which make use of the inherent creativity, such as creating and evolving simulated lifeforms (Sims, [16]), or designing radar antennas (DeJong et. al. [3]). There have also been some applications using GAs to evolve shape grammars in 2D (O'Neill et. al. [11]), or improve the structure of power pylons (Byrne et. al. [2]).

3 Approach

In this paper we use GAs as a method to stochastically explore all possible derivation trees for a given grammar and select those that best fit a given criteria.

The GA requires only minimal explicit knowledge about the rules of the grammar and can optimize towards any criteria that can be used to rank the derivation trees, and can provide many different viable solutions.

To illustrate our approach we focus on a grammar that creates spaceship models by accumulating geometric shapes, and use the volume of a target model as an optimization criteria. To ensure the volume of the target model is possible to reach, we generate it using the same production system and a fixed derivation tree, but any target volume could be used.

3.1 Genetic Algorithm

Key points for the functionality of GAs are the distinction between genome encoding (*genotype*) and the expression of the genomes in the problem space (*phenotype*), and genetic operators. Operators modify the chromosomes of individuals without necessarily having any information about how the modifications affect the solutions. This distinction allows the GA to operate on a variety of problems, but requires the definition of genome representation and genetic operators for each specific problem. In this chapter we specify the genome representation and the genetic operators, and give details about the implementation of the GA.

3.1.1 Genome Representation

The genome representation should encode complex operations in the problem space in a way that allows the GA to identify and combine building blocks for good solutions, while being as simple as possible, to keep the chromosomes manageable. At the same time, since the fitness evaluation of a given individual usually requires a translation of the genes into their expression, they should also be easy to decode.

For shape grammars, the traditional approach of using bit-strings of fixed size is somewhat limited, but the structure of the derivation tree facilitates these features, so we use it as our genome representation. This representation is similar to the ones used by [10, 16, 22], where a tree structure is encoded within a chromosome in various ways.

In our representation a single gene consists of a production symbol and it's parameters, and a reference to the parent gene within the tree structure. A vector of genes form a chromosome, which can be expressed as a derivation tree and used by a production system to generate a model.

This structure also allows for a variable length of the chromosomes and easy insertion of new genes into the chromosome without changing the already existing entries.

To allow the operators to keep the generated or modified chromosomes within the confines of a valid derivation tree, each possible type of gene is defined by a symbol descriptor (Fig. 1), which contains the possible child symbols, how likely they are to be generated during mutation and a description of the parameters. The information in these descriptors is inferred from the rules for the given grammar.

3.1.2 Crossover Operator

The crossover operator produces a viable pair of children given a pair of parents. The simplest form is the single-

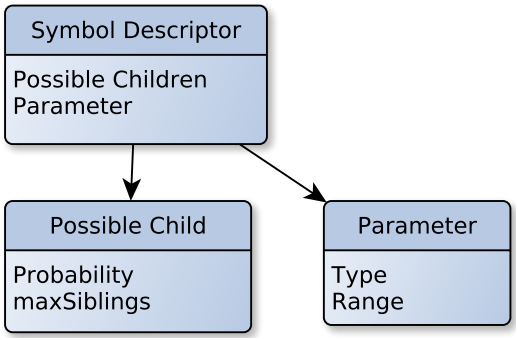

Figure 1: Structure of the symbol descriptor.

point-crossover, which selects a random crossover point in both parent-chromosomes and creates the offspring by swapping the genes after the crossover points. The number of possible different children given the same set of parents is limited by the number of valid crossover points.

Our operator uses a variation of the single-point-crossover operator, adapted to tree structures. It selects a random gene connection from the first parent, and chooses another random connection from all compatible connections in the second parent, and exchanges the genes at these connections. This operation is outlined in Fig. 2, and presented in more detail at Algorithm 1.

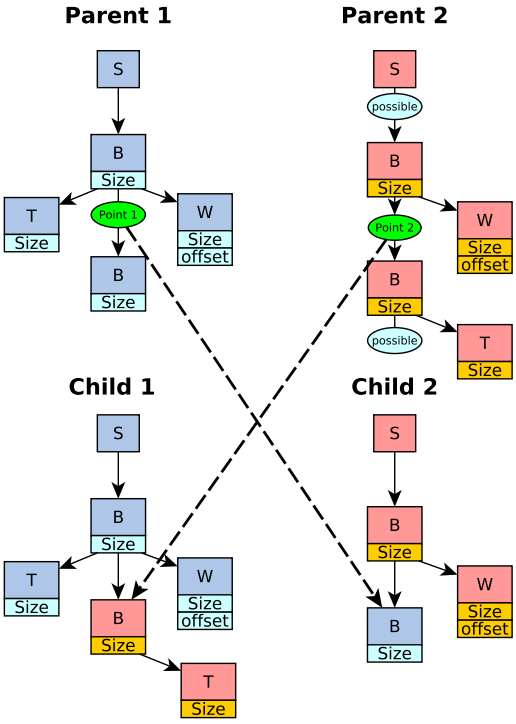

Figure 2: The crossover operator. For a random connection in the first parent, a fitting connection in the second parent is selected, and the offspring is generated.

3.1.3 Mutation Operator

Mutation allows the GA to explore the problem space outside the already existing population by introducing random changes in ways that are very unlikely to occur by using crossover operators alone. Our operator uses the following changes that arise intuitively from the tree structure of the chromosomes:

- **Grow:** Adds a suitable gene as child of a random gene and initializes the parameters. This doesn't replace already existing genes.
- **Cut:** Removes a random gene and all child-genes.
- **Permutate:** Change the parameter values of a random gene.

When the operator is applied, one of these changes is chosen at random.

3.1.4 Selection Methods

The selection method is one of the central parts of GAs, as it allows the algorithm to select good parents for the new generation, while at the same time denying bad solutions the chance to reproduce. This is usually done by selecting the individuals according to their fitness values, with some margin for error.

With a purely deterministic selection method the same parents would be selected again and again, thereby limiting the exploration of the problems space to the proximity of the fittest individual of the initial population, whereas a purely random method would lead to an entirely undirected exploration, which decreases the probability of finding a good solution considerably. There are several possible selection operators, as compared by [1], but the ones most widely used are roulette wheel selection and k-tournament selection.

Roulette Wheel selection calculates the probability p_i that the individual i is chosen such that it is proportional to it's fitness value f_i in relation to the overall fitness of the population:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{1}$$

The name stems from the informal description of the method as a roulette wheel, where the size of each possible spot relates to the fitness of the individual occupying it.

k-tournament selection consists of selecting k individuals at random from the population, and choosing the individual with the highest fitness value, i.e. letting them fight in a tournament. This method is quite fast, and the selection pressure can be increased by increasing the size of the tournament.

Algorithm 1 Crossover operator

```
procedure CROSSOVER(parent1, parent2)
  while tries < Max Retries do
    Point1 ← Random connection from parent1
    C ← SELECTPOSSIBLE(parent2, Point1)
    if C is empty then tries ← tries + 1
    else
      Point2 ← random connection from C
      child1 ← CLONEUNTIL(parent1, Point1.start)
      child1 ← CLONEFROM(parent2, Point2.end)
      child2 ← CLONEUNTIL(parent2, Point2.start)
      child2 ← CLONEFROM(parent1, Point1.end)
      return child1, child2
procedure SELECTPOSSIBLE(parent2, Point1)
  for all Connections C in parent do
    if C.end is possible Child of Point1.start then
      // Count siblings, including the current one
      S ← Number of siblings with type of C.end
      // Check if there can be more genes of this type
      Dc ← PossibleChild (Fig. 1) of C.end
      if DC.maxSiblings == S then
        connections.add(C)
      else
        P ← random percentage value
        if P < Replace probability then
          connections.add(C)
        else
          Cnew ← new connection
          Cnew.start ← C.start
          Cnew.end ← none
          connections.add(Cnew)
  return connections
```

3.1.5 Implementation Details

There is a large variety in the details of GAs, which differ slightly in each implementation. The variants used for this paper are as follows:

The Initial Population is created by repeatedly applying the grow-mutation operator to initially empty chromosomes. The GA converges faster if the chromosome-length of the initial population is comparable to the desired target, but the length is self-correcting to a large extent.

A new generation is created by selecting two individuals from the population, and either applying the crossover-operator or the mutation operator to both individuals.

Additionally we use **elitism**, whereby some individuals with the best fitness values are copied unchanged to the new generation, but are still used as parents for crossover. This ensures that the quality of the solution never decreases, and can also improve the quality of the solution, since good individuals are preserved and produce more offspring. But if the population size is too small, this can lead to stagnation.

3.1.6 Fitness function

The fitness function evaluates the quality of a single individual and assigns a fitness value to it. Since the GA only optimizes the fitness value, the used fitness function very much defines the visual quality of the solutions. It also should not be defined too restrictive, to allow the GA to explore non-optimal solutions. Additionally one has to pay attention to the complexity of the function, since the fitness calculation is often the most time consuming task of a GA. There are many possible different fitness functions for shape grammar, like evaluating the silhouette from a certain perspective, or several volume-based approaches, such as filling or avoiding a given volume.

We use a volume-based fitness function, where we generate the model using the derivation tree defined by a chromosome, and compare the volume of the model to a given target volume. The comparison is done by converting the generated model into voxels using a basic ray-based voxelisation method and counting the voxels. Then, with v_{target} as the number of voxels of the target volume, v_{inside} as the number of generated voxels that fall inside the target volume, $v_{outside}$ the number of voxels outside the target volume, and $v_{overlap}$ as the number of voxels that are self-overlapping within the generated model, the fitness value f is calculated with

$$f_{good} = step(v_{inside}, 0, v_{target}) \tag{2}$$

$$f_{bad} = step(v_{outside} + v_{overlap}, 0, 2 \cdot v_{target}) \tag{3}$$

$$f_{length} = step(l, l_{opt}, l_{max}) \tag{4}$$

$$f = \alpha \cdot f_{good} - \beta \cdot f_{bad} - \gamma \cdot f_{length} \tag{5}$$

with α, β, γ as weights. Furthermore l is the length of the chromosome, and l_{opt} and l_{max} are given parameters, since it has been shown by [17] that including the length of the chromosome in the fitness calculation is a good way to prevent it from growing considerably, which would increase the evaluation time.

Finally we use a $step$ - function to provide a normalization of the fitness value, based on the *smooterhstep* - function defined in [13].

$$step(x, min, max) = \begin{cases} 1 & \text{if} \quad x \geq max \\ 0 & \text{if} \quad x \leq min \\ 6t^5 - 15t^4 + 10t^3 & t = \frac{x-min}{max-min} \end{cases} \tag{6}$$

This is the most time-consuming step of our implementation, but by using an efficient implementation on the GPU and an variant of *CGA-Shape* previously used by [19], we were able to keep the calculation time manageable.

4 Evaluation

To evaluate the method presented in this paper, we implemented a framework using C++ and CUDA.

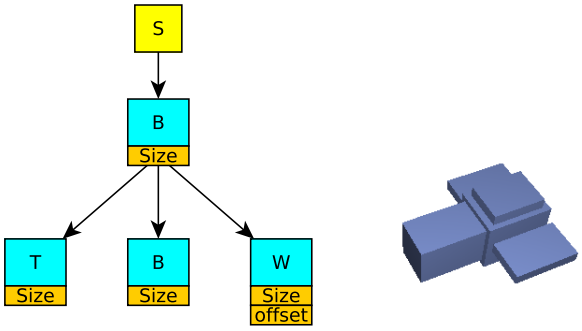

Figure 3: Basic symbols of the grammar and the generated model using fixed values for the parameters.

4.1 Spaceship Grammar

The implemented grammar is inspired by [14], and is designed to produce simple spaceships with wings. It produces axis-aligned boxes of varying sizes, and consists of a start symbol **S**, and three terminal symbols **B**, **W** and **T**. The rules are not explicitly required for this method, but can be outlined as follows:

$$\begin{array}{lcl} \mathrm{S} & \rightarrow & \mathrm{B} \\ \mathrm{B} & \rightarrow & \mathrm{BB} \mid \mathrm{BW} \mid \mathrm{BT} \mid \varepsilon \\ \mathrm{W} & \rightarrow & \mathrm{WW} \mid \varepsilon \\ \mathrm{T} & \rightarrow & \mathrm{TT} \mid \varepsilon \end{array}$$

The three basic symbols relate to the following shapes:

- **B**: central spaceship-body, attached to the parent symbol along the main axis.
- **W**: wings, mirrored at the main axis and attached along the secondary axis.
- **T**: top, attached along the third axis.

A basic example is shown in Fig. 3.

4.2 Parameter Selection

The selection of parameters for a GA is a very complicated task, since the parameters are interdependent, e.g. a high mutation rate can produce good results, but only if the population size is large enough. There have been various attempts to optimize this selection, such as using statistical models [4], or even using other optimization algorithms to find the best set of parameters, which introduces the problem of finding parameters for that algorithm. Since the execution time of our implementation is manageable, we were able to find a good set of parameters by changing one parameter at a time and comparing the results. The parameters during our evaluation are fixed to the following baseline, unless stated otherwise:

Population size:	50 individuals
Initial length:	10 symbols
Max. generations:	50
Elitism:	1 individual
First selection:	roulette wheel
Second selection:	k-tournament, size 10
Mutation prob:	30%
Mutation operator:	cut/grow/perm. uniform distr.
Crossover retries:	3

Due to the simplicity of the grammar, the crossover operator was able to produce an offspring reliably, with only about 0.2% of all cases requiring at most two tries.

The parameters for the fitness calculation do alter the look of the generated models, but do not alter the behavior of the GA significantly. They were set to the following values:

$$\alpha = 1 \qquad \beta = 0.8 \qquad \gamma = 0.2$$
$$l_{opt} = 40 \qquad l_{max} = 100$$

While the target can be an arbitrary volume, to ensure it is reachable we created it using the same grammar with a fixed derivation tree. The random number generator (RNG) was the uniform distributed mersenne-twister implementation provided by c++11 (mt19937). All the generated values are averaged over three discrete runs.

4.2.1 Selection method

The available selection methods are a roulette wheel selection, a k-tournament selection of size 10, and random selection. As shown in Fig. 4, using a semi-stochastic method for at least one parent produces better results than purely random selection, with tournament selection performing better. The best results were produces by a combination of tournament and roulette wheel selection, although initially tournament selection for both parents increases the fitness values slightly faster.

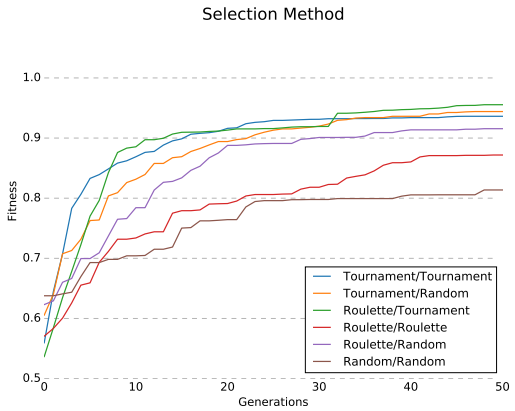

Figure 4: The fitness values with different selection methods.

4.2.2 Population size

Since the GA recombines parts of already existing solutions, a bigger population increases the chance to combine two good parts to create a better solution, and it also increases the probability that an individual already has a good fitness value from the beginning, therefore increasing the fitness of the solution immediately. Unfortunately, an increase in the population size also increases the execution time significantly, which requires finding a trade-off between speed and fitness. When increasing the population from 10 to 500 individuals, as shown in Fig. 5, the fitness increases as well, but after a size of 200 individuals the increase is negligible compared to the increased execution time.

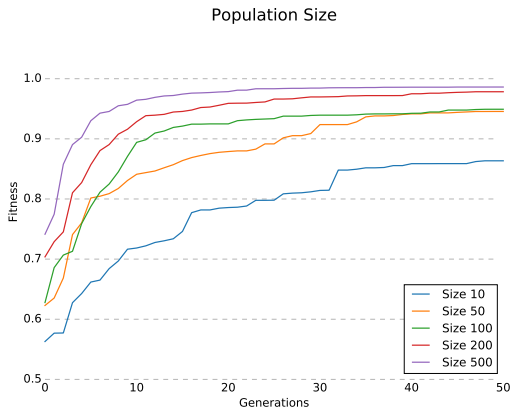

Figure 5: The fitness values with increasing population size.

4.2.3 Mutation probability

If the mutation rate is too low, the probability to produce beneficial changes is low as well, while a high mutation rate can introduce disadvantageous changes to already good solutions. This can be mitigated to some extend with elitism, which can introduce another set of problems. When the mutation rate is increased from 0% (only crossover) to 100% (only mutation) (shown in Fig. 6), the fitness increases as well, although the difference is only significant in later generations.

Using only mutation produces good results, but we found in further evaluation that the effect diminishes with a higher population size.

4.2.4 Elitism

Elitism allows the GA to explore the problem space surrounding good solutions by keeping them unchanged from one generation to the next, while still using them as parents. A small elitism rate in a large population can lead to a replacement of the elite in every turn, thus having no impact at all, while a large elitism rate can lead to stagnation. By changing the elitism rate from 0 to 45 individuals

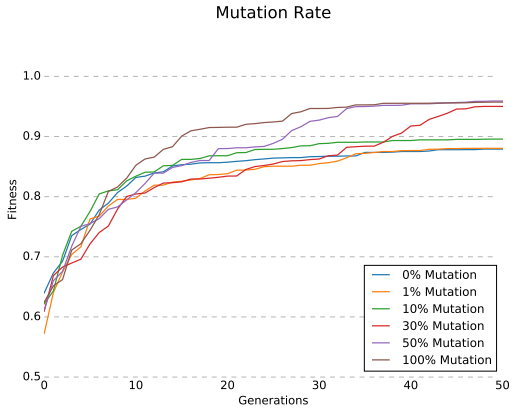

Figure 6: The fitness values with increasing mutation rate.

(90%), as shown in Fig. 7, we find that the use of 10 to 15 individuals (20 - 30%) produces the best results, although the impact is not very significant. It does, however, ensure a steadily increasing fitness value. We also found in further evaluations that the impact of elitism is highest with small populations, and diminishes with increasing population sizes. And, as expected, keeping a large part of the population as elite does decrease the quality of the result significantly, and also leads to periods of stagnation.

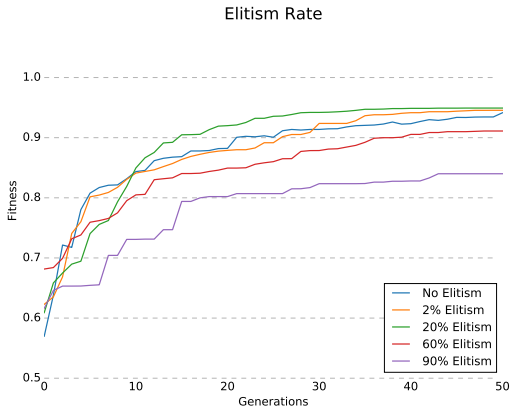

Figure 7: The fitness values with increasing elitism levels.

4.2.5 Initial length

The length of the initial population was increased from 10 symbols up to 100 symbols (Fig. 8). Due to the influence of the length on the generated model, this significantly alters the fitness of the initial population, which in turn impacts the performance of the algorithm. But we found that the impact diminishes with increased generations.

4.2.6 Final parameters

Overall we found that the population size and the use of any selection method other than random selection have the

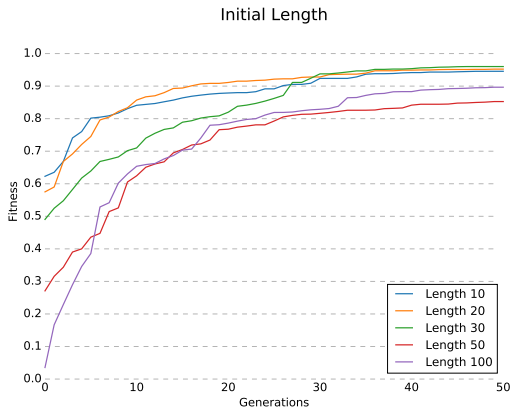

Figure 8: The fitness values with an increasingly complicated initial population.

largest impact on the quality of the result, while other parameters have the most impact during the first few generations. Thus we were able to generate good results using the following parameters:

Population size:	200 individuals
Initial length:	20 symbols
Max. generations:	50
Elitism:	30 individuals
First selection:	roulette wheel
Second selection:	*k*-tournament, size 10
Mutation prob:	30%
Mutation operator:	cut/grow/perm. uniform distr.

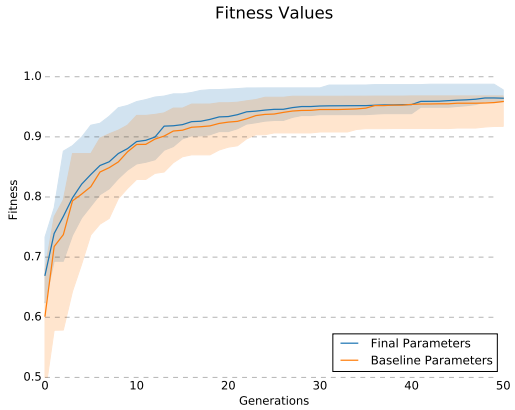

Figure 9: The fitness values for the baseline and final parameters.

The results of the baseline and the final parameters are very similar (shown in Fig. 10), since the baseline parameters already produce good results, but the final parameters tend to perform more reliably. But because the bigger population creates a longer execution time, the parameters can be optimized with regards to time for specific targets.

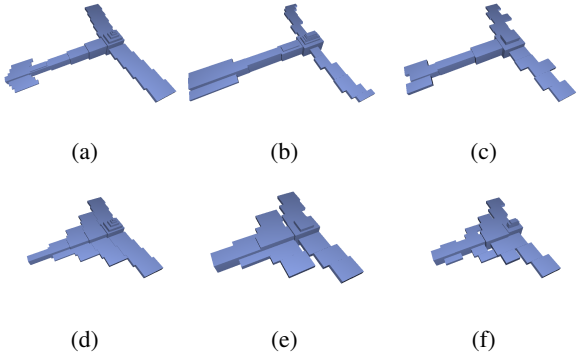

Figure 10: Targets and results of various runs: *a* and *d* are targets, *b* and *e* were generated using the baseline parameters, and *c* and *f* were generated with the final parameters.

Population	**Evaluation (avg)**	**Crossover (avg)**	**total**
50	0.25sec	2ms	12.38sec
200	0.9sec	11ms	45.63sec

Table 1: Average evaluation and reproduction times per generation, and the total duration for one run.

4.3 Performance

The results of the performance evaluation are shown in table 1. Most time is spent calculating the fitness values, while the time for generating a new generation is minimal. But since most of the evaluation time is spent on generating the model from the derivation tree, this time could be improved by moving the production system to the GPU, and exploiting the inherent parallelism of GAs.

These values were achieved on an Intel Core i5-5200U CPU @ 2.20GHz with 8 GB of RAM and a Nvidia Geforce 840M.

5 Conclusion and Future Work

We presented a genome representation and genetic operators that are suitable for an application of GAs to control the derivation tree for shape grammars. Furthermore we demonstrated the basic viability of this approach by presenting the implementation for a specific grammar for simple spaceships and a volume-based fitness function, and evaluated the influence of the parameters required for GAs.

A clear opportunity for future work is the evaluation of this method with different, more complicated shape grammars, since GAs generally tend to perform worse when the complexity of the problem space increases.

Also is the volume-based fitness function very restrictive and limits the creative capabilities of GAs, therefore a different approach for the fitness calculation might be

preferable, such as a image-based evaluations, as used successfully by [14, 22].

Furthermore could an implementation of the algorithm on the GPU improve the performance considerably, since GAs and shape grammars are inherently parallel in nature.

References

[1] T. Blickle and L. Thiele. A Comparison of Selection Schemes Used in Evolutionary Algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[2] J. Byrne, M. Fenton, E. Hemberg, J. McDermott, and M. O'Neill. Optimising complex pylon structures with grammatical evolution. *Information Sciences*, 316:582–597, 2015.

[3] C. M. De Jong Van Coevorden, A. R. Bretones, M. F Pantoja, F. J. García Ruiz, S. G. García, and R. G. Martín. GA design of a thin-wire bow-tie antenna for GPR applications. *IEEE Transactions on Geoscience and Remote Sensing*, 44(4):1004–1009, 2006.

[4] O. François and C. Lavergne. Design of evolutionary algorithms - A statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.

[5] J. H Holland. *Adaptation in natural and artificial systems*. 1992.

[6] L. Krecklau, J. Born, and L. Kobbelt. View-dependent realtime rendering of procedural facades with high geometric detail. In *Computer Graphics Forum*, volume 32, pages 479–488. Wiley Online Library, 2013.

[7] A. Lindenmayer. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of theoretical biology*, 18(3):300–315, 1968.

[8] P. Merrell and D. Manocha. Model synthesis: A general procedural modeling algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):715–728, 2011.

[9] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Transactions on Graphics*, 25(3):614, 2006.

[10] E. Murphy, M. O'Neill, E. Galván-López, and A. Brabazon. Tree-adjunct grammatical evolution. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[11] M. O'Neill, J. M. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg. Shape grammars and grammatical evolution for evolutionary design. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*, page 1035, 2009.

[12] Y. I. H. Parish and P. Müller. Procedural Modeling of Cities. *28th annual conference on Computer graphics and interactive techniques*, (August):301–308, 2001.

[13] K. Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. ACM.

[14] D. Ritchie, B. Mildenhall, and P. Goodman, N. D .and Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Transactions on Graphics (TOG)*, 34(4):105, 2015.

[15] M Schwarz and P. Müller. Advanced procedural modeling of architecture. *ACM Transactions on Graphics*, 34(4 (Proceedings of SIGGRAPH 2015)):107:1–107:12, August 2015.

[16] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[17] T. Soule, J. A. Foster, and J. Dickinson. Code Growth in Genetic Programming. *GECCO '96 Proceedings of the 1st annual conference on genetic and evolutionary computation*, pages 215–223, 1995.

[18] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. Inverse procedural modelling of trees. In *Computer Graphics Forum*, volume 33, pages 118–131. Wiley Online Library, 2014.

[19] M. Steinberger, M. Kenzel, B. Kainz, J. Mueller, W. Peter, and D. Schmalstieg. Parallel generation of architecture on the GPU. *Computer Graphics Forum*, 33(2):73–82, 2014.

[20] G. N. Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars and Aesthetic Systems.* PhD thesis, 1975.

[21] G. N. Stiny. Spatial Relations and Grammars. *Environment and Planning B: Planning and Design*, 9(1):113–114, mar 1982.

[22] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. Metropolis procedural modeling. *ACM Transactions on Graphics*, 30(2):1–14, 2011.

[23] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant Architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.